# Generation of structured hexahedral meshes in volumes with holes

E. Ruiz-Gironés[1] and J. Sarrate[2]

[1] Laboratori de Càlcul Numèric (LaCàN),
Departament de Matemàtica Aplicada III,
Universitat Politècnica de Catalunya,
Jordi Girona 1-3, E–08034 Barcelona, Spain
Tel. 34-93-401 79 59, Fax: 34-93-401 18 25
e-mail: eloi.ruiz@upc.edu

[2] Laboratori de Càlcul Numèric (LaCàN),
Departament de Matemàtica Aplicada III,
Universitat Politècnica de Catalunya,
Jordi Girona 1-3, E–08034 Barcelona, Spain
Tel. 34-93-401 69 11, Fax: 34-93-401 18 25
e-mail: jose.sarrate@upc.edu

**Abstract**

The submapping method is one of the most used techniques to generate structured hexahedral meshes. This method splits the geometry into pieces logically equivalent to an hexahedron. Then, it meshes each patch keeping the mesh compatibility between pieces by solving an integer linear problem. The quality of the final discretization is governed by the objective function that defines the linear problem. Thus, in this work we propose a new objective function that better distributes the number of intervals among the edges of the geometry. In addition, special procedures have to be developed in order to apply the submapping method to volumes with holes. This article also presents two original contributions to efficiently mesh geometries that contain holes. Finally, it presents several numerical examples that show the applicability of the developed algorithms.

**keywords** Mesh generation, submapping, structured hexahedra, linear programming,p transfinite interpolation, multiply connected geometries.

# 1 Introduction

The Finite Element Method is one of the most important techniques in applied sciences and technology. However it is hampered by the need to generate a discretization adapted to the geometry and to the prescribed distribution of element sizes. Several fast and robust algorithms have been developed to generate triangular and tetrahedral meshes (see [1, 2] for a detailed survey). In these methods local connectivity modifications are a crucial step. Nevertheless, in hexahedral meshes the connectivity modifications propagate through the mesh. In this sense, hexahedral meshes are more constrained, and therefore, much more difficult to generate. Several algorithms have been developed to generate unstructured hexahedral meshes for a given geometry: grid-based methods [3, 4], medial surface methods [5, 6] advancing front techniques [7] or dual methods [8, 9]. Although all the previous algorithms to generate hexahedral meshes have their strengths and weaknesses, a general and fully automatic hexahedral mesh generation algorithm is still not available. Moreover, further research is still needed to develop a general purpose algorithm that generates high quality hexahedral elements for any arbitrary geometry. Therefore, special attention has been focused on existing algorithms that decompose the entire geometry into several simpler pieces (assembly models). These smaller volumes can be easily meshed by well-known methods that obtain an outstanding performance in these simpler volumes, such as: mapping [10], submapping [11, 12], and sweeping [13, 14, 15, 16, 17].

The accuracy of a FEM simulation is directly related to the quality of the discretization used. In this sense, structured meshes are still preferred in a wide range of simulations where a strict alignment of elements is required by the analysis. For instance, boundary layers in computational fluid dynamics or composites in structural dynamics. One of the most important techniques to generate structured hexahedral meshes is the submapping algorithm [11, 12]. This method relies on a geometric decomposition of the domain into patches logically equivalent to an hexahedron. Then, each patch is meshed separately using a standard structured mesh generation algorithm. In our implementation, we have used the transfinite interpolation method (TFI), see [1] for details. The mesh compatibility between patches is previously imposed by solving an integer linear problem (ILP). The quality of the obtained mesh is governed by the objective function of the integer linear problem. For this reason, we propose a new objective function that better distributes the elements between the edges of the geometry.

The submapping method has been extensively used to mesh simply connected *blocky* geometries. Therefore, special algorithms have to be developed in order to discretize volumes with holes. In this work, we present two original contributions that allow to mesh geometries that contain holes. The first one is devoted to discretize volumes with inner voids. We deduce a new procedure that automatically

connects inner boundaries with the outer boundary. In this way, we transform the initial volume into a simply connected domain. Then, we can apply the standard submapping algorithm to obtain the final hexahedral mesh.

The second contribution is focused on volumes with through holes. If through holes are present, the solution of the integer linear problem does not guarantee the mesh compatibility between patches. To overcome this drawback, we propose a new algorithm based on a graph representation of the geometry. This algorithm has two main advantages. On the one hand, it verifies compatibility between patches by including the necessary constraints to the integer linear problem. On the other hand, it can also be applied to simply connected domains and geometries with inner voids. Moreover, in these two cases it reduces the number of constraints of the integer linear problem.

The remainder of the paper is organized as follows. In Section 2 we briefly detail the basis of the submapping method, and we introduce the new objective function for the ILP. In Section 3 we present two original algorithms to mesh volumes with inner and through holes. Finally, in Section 4 we present several examples to illustrate the applicability of the proposed algorithms.

# 2 The submapping method

In order to generate a structured mesh of hexahedral elements, we will assume that there exists a representation of the geometry in which every edge is parallel to the coordinate axis. We define this representation as the computational domain, whereas the initial geometry is defined as the physical domain, see [11, 12] for details. The coordinate axis of the computational domain are denoted by $I$, $J$, and $K$. Figure 1 shows the physical and computational domain of a simple geometry.

## 2.1 Vertex, edge and face classification

For each face of the geometry, the vertices are classified according to the angle defined by their adjacent edges. Since the angles are an integer multiple of $\pi/2$, a vertex can be classified as *side* (the angle is 0), *end* (the angle is $\pi/2$), *reversal* (the angle is $\pi$) or *corner* (the angle is $3\pi/2$).

The edges of the geometry are classified in two ways. The first is local, in the sense that it is different for each face in which the edge is contained. Since each face is submappable, it defines a local 2D computational domain, $(I, J)_f$. Thus, an edge can be classified as $I_f^{\pm}$ or $J_f^{\pm}$ according to the direction that it takes in the local computational domain of face $f$. More specifically, given a face, $f$, we

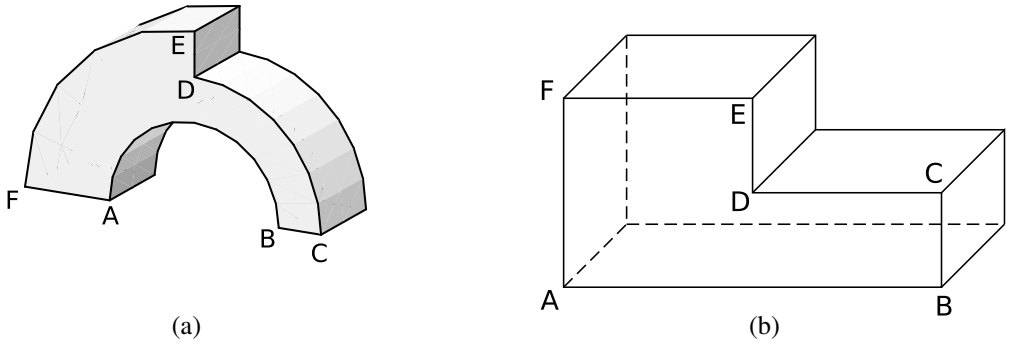



Figure 1: Physical and computational domain of a given volume. (a) Physical domain. (b) Computational domain.

define the following sets of edges:

$$\begin{cases} I_f^+ & \text{edges of face } f \text{ in the } I^+ \text{ direction,} \\ I_f^- & \text{edges of face } f \text{ in the } I^- \text{ direction,} \\ J_f^+ & \text{edges of face } f \text{ in the } J^+ \text{ direction,} \\ J_f^- & \text{edges of face } f \text{ in the } J^- \text{ direction.} \end{cases}$$

The second way to classify edges is global. They are classified according to the angle that define their adjacent faces. Recall that these angles are, approximately, an integer multiple of $\pi/2$. For this reason, an edge can be classified as *side* (the angle is 0), *end* (the angle is $\pi/2$), *reversal* (the angle is $\pi$) or *corner* (the angle is $3\pi/2$).

The faces of the geometry are classified according to their normal in the computational domain. Consequently, a face can be classified as $I^{\pm}$, $J^{\pm}$ or $K^{\pm}$. Figure 2 shows vertex, edge and face classification of a simple geometry.

## 2.2 Interval assignment

In order to generate a structured mesh we have to compute the number of divisions, $n_e$, of each edge $e$. To this end, we first assign a target number of divisions, $N_e$, to every edge. Then, we propose to minimize the new objective function:

$$\sum_e \omega_e |n_e - N_e| + M - m, \tag{1}$$

where $M$ and $m$ are the maximum and minimum differences between the target number of intervals and the computed number of intervals, and $\omega_e$ is a weight

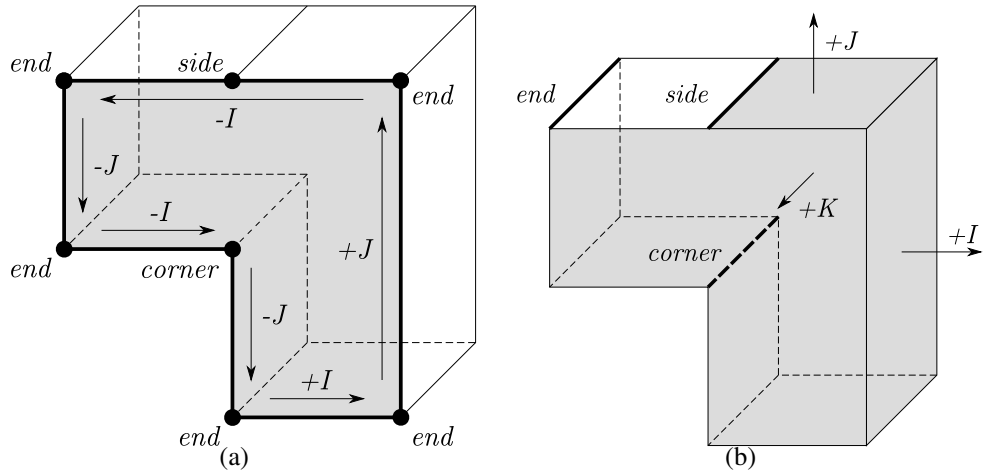



Figure 2: Vertex, edge and face classification. (a) Node classification and edge local classification for the front face. (b) Edge global classification and face classification.

that controls the cost of increase or decrease the number of intervals of edge $e$. We define this weight as $\omega_e = 1/l(e)$, where $l(e)$ is the length of edge $e$ in the physical domain. Note that the unknowns of objective function (1) are $n_e$, $M$ and $m$.

Although equation (1) is not a linear function we can modify it to obtain a linear one. Each absolute value is decomposed into the sum of two positive variables:

$$|n_e - N_e| = D_e + d_e, \qquad \text{for all edges,} \tag{2}$$

where $D_e$ and $d_e$ are the positive and negative differences between $n_e$ and $N_e$, respectively. That is:

$$D_e = \begin{cases} n_e - N_e & \text{if} \quad n_e - N_e > 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad d_e = \begin{cases} N_e - n_e & \text{if} \quad n_e - N_e < 0 \\ 0 & \text{otherwise} \end{cases}.$$

In addition a new constraint is introduced:

$$D_e - d_e = n_e - N_e, \qquad \text{for all edges.} \tag{3}$$

Note that equation (2) and constraint (3) allow to eliminate the absolute value in the objective function (1). Therefore, to determine a compatible subdivision for

the edges of the geometry we solve the following integer linear problem:

$$\begin{cases} \min \sum_e \omega_e (D_e + d_e) + M - m, & \\ \text{constrained to:} & \\ \sum_{e \in I_f^+} n_e = \sum_{e \in I_f^-} n_e, & \text{for all faces,} \\ \sum_{e \in J_f^+} n_e = \sum_{e \in J_f^-} n_e, & \text{for all faces,} \\ D_e - d_e = n_e - N_e, & \text{for all edges,} \\ D_e, d_e \geq 0, & \text{for all edges,} \\ M \geq n_e - N_e, & \text{for all edges,} \\ m \leq n_e - N_e, & \text{for all edges.} \end{cases} \tag{4}$$

The first novelty of our approach resides in the modification of the objective function in (4). Reference [18] proposes to solve several linear problems of decreasing size to obtain the number of edge intervals. Moreover, the objective function to minimize is the maximum difference between the target number of intervals and the actual number of intervals. Instead, our approach takes into account the global change of the number of intervals, $\sum_e \omega_e (D_e + d_e)$, the maximum difference, $M$, and the minimum difference, $m$.

The solution of the integer linear problem (4) provides a number of edge intervals that leads to structured hexahedral mesh. This kind of problems can be solved using the branch & bound method [19]. In our implementation, we used the lpsolve libray [20] to solve integer linear problem (4). Once the problem is solved, we can mesh each face using surface submapping and then construct the computational domain. Recall that, in the computational domain, we know the direction of each edge for all faces, and the normal vector of each face. Therefore, we only need the edge lengths to construct the computational domain. Thus, we define the length of an edge in the computational domain as its number of intervals.

## 2.3 Geometry decomposition

The decomposition of the geometry is performed in the computational domain. To decompose the domain, we will use cutting surfaces. We impose that cutting surfaces start at edges classified as *corner* or *reversal*. In addition, we also impose that cutting surfaces are parallel to the coordinate axis in the computational domain. Among the possible cutting surfaces, we will choose the one with shortest perimeter. When the geometry is decomposed into two parts, we iterate the process recursively in both parts until there are no edges classified as *corner* nor

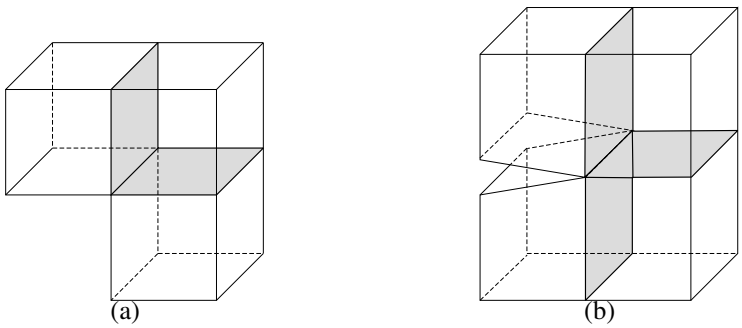



Figure 3: Possible cutting surfaces for *corner* and *reversal* edges. (a) *Corner* edge. (b) *Reversal* edge.

*reversal*. Figure 3 shows the possible cutting surfaces for *corner* and *reversal* edges.

When there is a patch with no edges classified as *corner* nor *reversal*, it can be meshed using a classical mapping algorithm, since it is logically equivalent to an hexahedron. Algorithm 1 summarizes the proposed submapping algorithm for simply connected geometries.

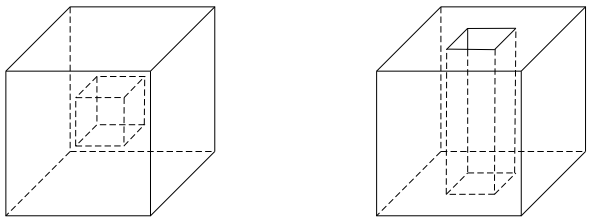
**Algorithm 1** Submapping method

**Ensure:** aMesh

1: **function** SUBMAPPING(aVolume)
2: Vertex classification
3: Edge classification
4: Face classification
5: Interval assignment: solution of ILP (4)
6: Boundary discretization
7: Geometry decomposition
8: Patch discretization
9: **end function**

# 3 Volumes with holes

Algorithm 1 is only valid for simply connected volumes. From the topological point of view, the holes of a volume can be classified as: inner voids (Figure 4(a)), and through holes (Figure 4(b)). In the first case, the boundary of the volume is composed by more than one closed set of faces. Thus, once the outer boundary is positioned in the computational domain, special algorithms have to be developed



(a) (b)

Figure 4: Multiply connected geometries. (a) Geometry with an interior void. (b) Geometry with a through hole.

to locate the inner boundaries. In the second case, if a volume contains through holes, two problems arise. On the one hand, one cutting surface does not split the geometry into two parts. Consequently, it is necessary to compute additional cutting surfaces until the volume is divided into two separated parts. On the other hand, the solution of integer linear problem (4) does not ensure that the interval assignment allows the generation of an hexahedral mesh. The reason is that there are missing constraints in problem (4).

## 3.1 Inner voids

Special procedures have to be developed in order to apply Algorithm 1 to volumes that contain inner holes. This kind of volumes are composed by one set of faces that define the outer boundary and several sets of faces that define inner boundaries. Once the outer boundary is placed in the computational domain, we need additional information to properly locate the inner boundaries. Therefore, we propose to convert the geometry into a simply connected one by connecting inner boundaries with the exterior boundary. To this end, we will use virtual surfaces in the physical domain.

Algorithm 2 details the proposed method to mesh volumes with inner holes. Note that Algorithm 2 checks if the volume contains inner voids. The computational cost of this operation is extremely low since geometrical engines usually include this information. In our implementation we have used OpenCascade [21].

Given a geometry with inner voids, see Figure 5(a), steps 2 to 6 of Algorithm 2 provide a quadrilateral mesh of the volume boundary, see Figure 5(c). The proposed algorithm to connect inner boundaries with the outer boundary is composed by four steps:

(i) We select a quadrilateral face on an inner boundary and another face on the exterior boundary and proceed to remove them from the mesh (Figure 5(c)).

**Algorithm 2** The submapping algorithm for volumes with inner holes

**Ensure:** aMesh

```
1: function SUBMAPPING(aVolume)
2:     Vertex classification
3:     Edge classification
4:     Face classification
5:     Interval assignment: solution of ILP (4)
6:     Boundary discretization
7:     if Volume has inner holes then
8:         Connect inner boundaries with outer boundary
9:     end if
10:    Geometry decomposition
11:    Patch discretization
12:    if Volume has inner holes then
13:        Mesh the volume defined by the virtual surfaces
14:    end if
15: end function
```

The selection of these faces will be discussed later.

(ii) We connect the selected faces using a virtual (*tubular*) surface. The exterior boundary is updated by adding the virtual surface and the inner hole.

(iii) Iterate (i)-(ii) steps until all inner holes are connected to the outer boundary. Note that an inner hole can be connected to another inner hole that was previously connected to the outer boundary since the outer boundary was updated in step (ii).

(iv) We mesh the boundary of the virtual surfaces that we have created in steps (ii) and (iii), see Figure 5(d).

Once the geometry is simply connected, Algorithm 2 proceeds to mesh the new volume by decomposing the geometry using cutting surfaces and meshing separately each patch, see Figure 5(e). Finally, we have to mesh the inner part of the tubs defined by the virtual surfaces that we have previously created, see Figure 5(f).

The first step in the previous algorithm is to choose the cap faces for the tubs defined by the virtual surfaces that connect inner holes with the outer boundary. From the quadrilateral mesh of the boundary, we generate a constrained Delaunay tetrahedralization of the volume. In our implementation, this tetrahedralization is generated using the Tetgen library [22]. Figure 6 shows a constrained Delaunay

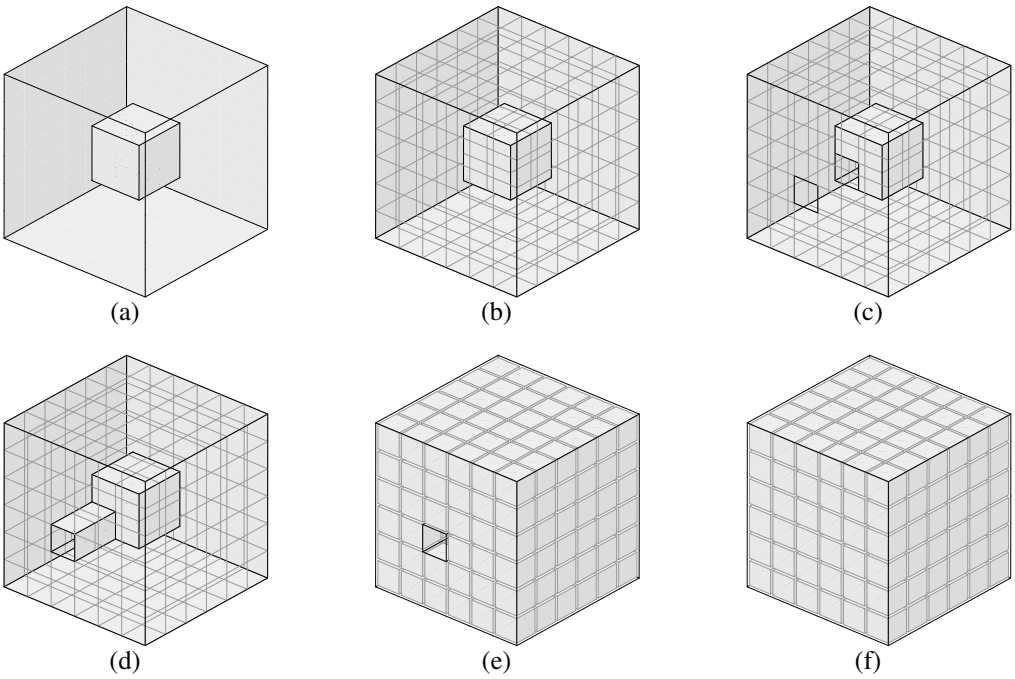



Figure 5: Multiply connected geometry converted into simply connected. (a) Volume with an inner hole. (b) Submapping boundary mesh. (c) Cap faces for the virtual surface. (d) Simply connected geometry. (e) Mesh generated in the new volume. (f) Final mesh.

tetrahedralization of a volume with an inner hole. The edges of this tetrahedralization will be used to select the faces to connect. Among all edges, we select the edge that connects the exterior boundary with an interior boundary and provides the *best* angles. That is, the angles are, approximately, an integer multiple of $\pi/2$. If more than one edge can be selected, then we will choose the shortest one.

The adjacent faces of the selected edge will be the candidates to be connected. Among all these faces, we will select the two of them that provide to the virtual surface the best angles. That is, the angles between the cap surfaces and the edges that connect them have to be, approximately, $\pi/2$. If more than one pair of faces can be selected we will pick the pair that are the closest each other. Note that virtual surfaces do not belong to the volume boundary. Therefore, they can be moved if a smoothing algorithm is applied later in order to improve the mesh quality.

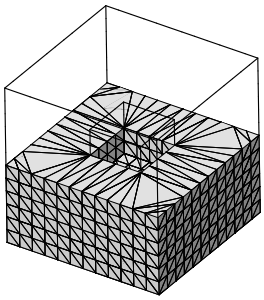


Figure 6: Constrained Delaunay tetrahedralization of a multiply connected volume.

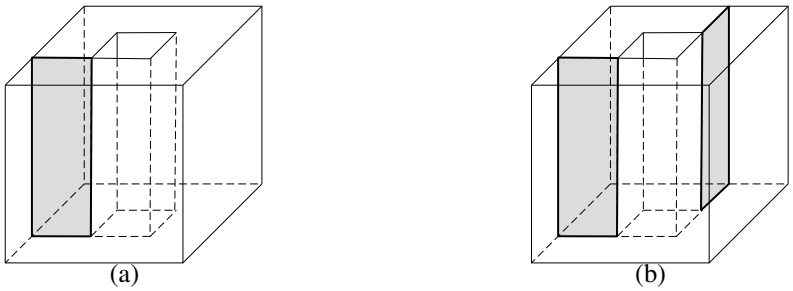



Figure 7: Cutting surfaces generated in a volume with a through hole. (a) First cutting surface. (b) Second cutting surface.

## 3.2 Through holes

Two problems arise when meshing volumes that contains through holes using the algorithm described in Section 2. The first of them is that a cutting surface may not divide the volume into two parts. Hence, it is necessary to compute additional cutting surfaces in order to split the volume into two parts. Figure 7 shows a volume with a through hole in which a cutting surface is not sufficient to divide the geometry into two parts. In this example, we need an additional cutting surface in order to split the volume.

The second problem is related to the interval assignment. The integer linear problem (4) may not provide compatible solutions if through holes are present. The reason is that there are missing constraints. For instance, consider the example presented in Figure 8. In this example, we have to impose an equal number of intervals for the selected edges. However, this constraint is missed if we construct the integer linear problem (4) face by face.

In this work, we propose to modify the construction of the integer linear prob-

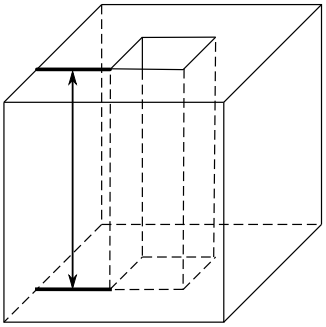


Figure 8: Interval assignment in a volume with a through hole.

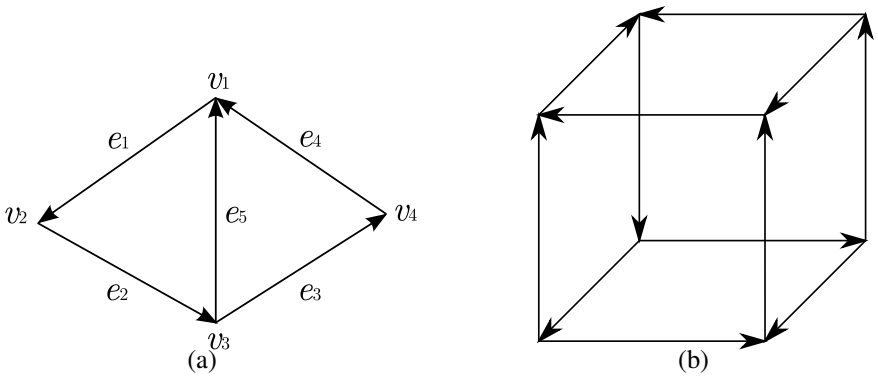



Figure 9: Two examples of directed graphs. (a) Directed graph defined by four vertices and five edges. (b) Directed graph defined by eight vertices and twelve edges, that is, a cube.

lem (4) by imposing a compatible number of intervals to every closed path of edges. To this end, we will consider a directed graph.

**Definition 1.** ***Directed graph.*** *A directed graph is a set of vertices, $V$, and a set of edges, $E$, such that each edge connects two vertices in such a way that it has an initial vertex and a final vertex. We denote the directed graph by $G = (V, E)$.*

Figure 9(a) presents a directed graph in which edge $e_1$ starts at vertex $v_1$ and ends at vertex $v_2$. In addition, it is worth to notice that a geometry can be considered as a directed graph. For instance, Figure 9(b) shows a cubic geometry represented as a directed graph. See reference [23] for more details.

**Definition 2.** ***Free group of vertices.*** *Given a directed graph, $G = (V, E)$, the free group of vertices is defined as the Cartesian product $\mathbb{V} = \mathbb{Z}^p$, where $n$ is the*

*number of vertices of the graph. An element* $\mathbf{v} \in \mathbb{V}$ *is expressed as a formal sum* $\mathbf{v} = \sum_{i=1}^{p} d_i v_i$*, where* $d_i \in \mathbb{Z}$ *and* $v_i$ *are the vertices of the graph. An alternative way to express this element is* $\mathbf{v} = (d_1, \ldots, d_p)$*. The dimension of* $\mathbb{V}$ *is* $p$*.*

*Given* $\mathbf{v} = (d_1, \ldots, d_p), \mathbf{v}' = (d'_1, \ldots, d'_p) \in V$ *the sum is defined as* $\mathbf{v}_{sum} = \mathbf{v} + \mathbf{v}' = (d_1 + d'_1, \ldots, d_p + d'_p)$*.*

**Definition 3.** ***Sample of vertices.*** *A sample of vertices is defined as an element* $\mathbf{v} = (d_1, \ldots, d_p)$ *of the free group of vertices,* $\mathbb{V}$*. Coefficients* $d_i$ *can be interpreted as the number of times the vertex* $v_i$ *appears in the sample* $\mathbf{v}$*. Note that the set of vertices,* $V$*, is the sample of vertices* $(1, \ldots, 1)$*.*

For instance, given the directed graph presented in Figure 9(a), let $\mathbf{v} = (1, 0, 0, -1)$ be an element of $\mathbb{V}$. $\mathbf{v}$ is a set of vertices in which: $v_1$ appears one time; $v_2$ and $v_3$ do not appear; and $v_4$ appears minus one times.

**Definition 4.** ***Free group of edges.*** *Given a directed graph,* $G = (V, E)$*, the free group of edges is defined as the Cartesian product* $\mathbb{E} = \mathbb{Z}^q$*, where* $q$ *is the number of edges of the graph. The elements of* $\mathbb{E}$ *are expressed as a formal sum* $\mathbf{e} = \sum_{i=1}^{q} c_i e_i$ *or alternatively as* $\mathbf{e} = (c_1, \ldots, c_q)$*. The dimension of* $\mathbb{E}$ *is* $q$*.*

*Given* $\mathbf{e} = (c_1, \ldots, c_q), \mathbf{e}' = (c'_1, \ldots, c'_q) \in E$*, the sum is defined as* $\mathbf{e}_{sum} = (c_1 + c'_1, \ldots, c_q + c'_q)$*.*

**Definition 5.** ***Path of edges.*** *A path of edges is defined as an element* $\mathbf{e} = (c_1, \ldots, c_q)$ *that belongs to* $\mathbb{E}$*. Coefficients* $c_i$ *are the number of times that edge* $e_i$ *is traveled. If the coefficient is negative, the corresponding edge is traveled in the opposite sense.*

For instance, given the graph presented in Figure 9(a), let $\mathbf{e} = (0, 0, 1, 1, -1)$. $\mathbf{e}$ is a path of edges in which: $e_1$ and $e_2$ are not traveled; $e_3$ and $e_4$ are traveled once in their original sense; and $e_5$ is traveled once in its opposite sense.

**Definition 6.** ***Boundary linear operator.*** *The boundary linear operator,* $\partial$*, is defined as:*

$$\begin{array}{rccc} \partial: & \mathbb{E} & \longrightarrow & \mathbb{V} \\ & \mathbf{e} & \longmapsto & \partial(\mathbf{e}) = \mathbf{v}, \end{array} \tag{5}$$

*where*

$$\partial(\mathbf{e}) = \partial\left(\sum_{i=1}^{q} c_i e_i\right) = c_1\partial(e_1) + \ldots + c_q\partial(e_q),$$

*and*

$$\partial(e_i) = s(e_i) - f(e_i), \qquad \textit{for } i = 1, \ldots, q,$$

*being* $s(e_i)$ *and* $f(e_i)$ *the starting and final vertices of* $e_i$*, respectively.*

For instance, in the example presented in Figure 9(a), let $\mathbf{e} = e_1 + e_2$. Then,

$$\partial(\mathbf{e}) = \partial(e_1 + e_2) = \partial(e_1) + \partial(e_2) = (v_1 - v_2) + (v_2 - v_3) = v_1 - v_3.$$

That is, the path defined by $\mathbf{e}$, starts at vertex $v_1$ and ends at vertex $v_3$.

**Remark 1.** *Operator $\partial$ can be expressed as a matrix, $\mathbf{B}$, in such a way that $\partial(\mathbf{e}) = \mathbf{Be}$. The coefficients of matrix $\mathbf{B}$ are defined as:*

$$\begin{cases} \mathbf{B}_{ij} = 1 & \iff & s(e_j) = v_i, \\ \mathbf{B}_{ij} = -1 & \iff & f(e_j) = v_i, \\ \mathbf{B}_{ij} = 0 & & \textit{otherwise}. \end{cases} \tag{6}$$

For instance, the matrix of the operator $\partial$ for the example presented in Figure 9(a) is:

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 \end{pmatrix}.$$

Note that the columns of matrix $\mathbf{B}$ represent the initial and final vertices of each edge.

In the previous example, we took $\mathbf{e} = e_1 + e_2 = (1, 1, 0, 0, 0)$. We have seen that $\partial(\mathbf{e}) = v1 - v3 = (1, 0, -1, 0)$. Using matrix $\mathbf{B}$ we should obtain the same result.

$$\mathbf{Be} = \begin{pmatrix} 1 & 0 & 0 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}.$$

**Definition 7** (Closed path of edges)**.** *A path of edges, $\mathbf{e}$, is closed if, and only if,*

$$\partial(\mathbf{e}) = \mathbf{Be} = \mathbf{0}. \tag{7}$$

*That is, a closed path of edges has no boundary.*

For instance, consider the graph presented in Figure 9(a). Let $\mathbf{e} = e_1 + e_2 + e_5 = (1, 1, 0, 0, 1)$ be a path of edges. It is a closed path since $\partial(\mathbf{e}) = 0$.

Thus, according to equation (7), closed paths belong to the kernel of matrix $\mathbf{B}$. Since closed paths are the kernel of a linear operator, it is straightforward to prove that they have the two following properties:

(i) Closed paths are a sub-group of $\mathbb{E}$. Therefore, two closed paths can be added, and the result is a closed path.

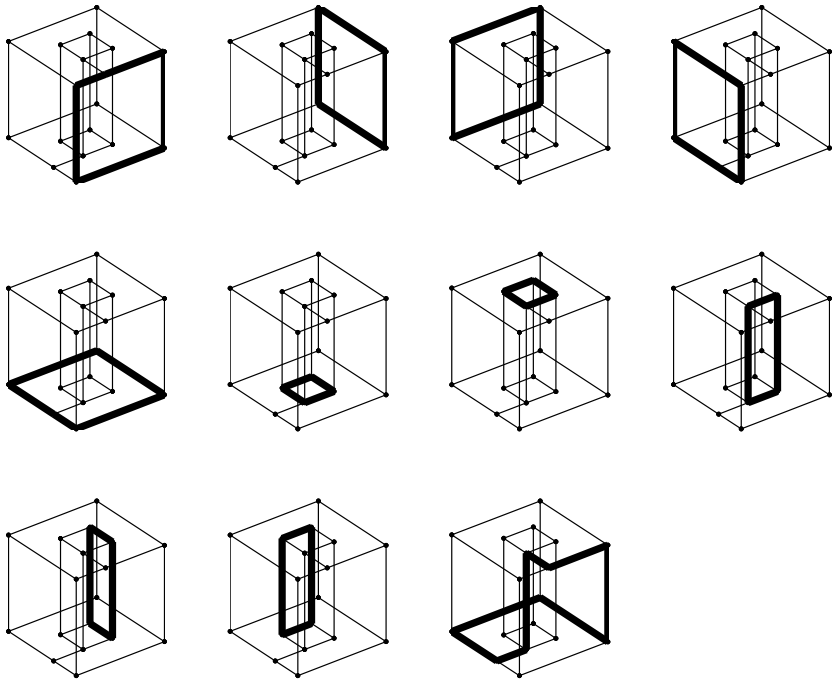


Figure 10: Basis of closed paths for a multiply connected geometry.

(ii) There exists a basis of closed paths such that every closed path is uniquely defined as a linear combination of the elements of the basis. Moreover, the coefficients of the linear combination are all integer.

Closed paths are the basic ingredient of the proposed algorithm to properly assign a compatible number of intervals to every edge. In fact, we will prove (see Proposition 2) that it is only necessary to impose the interval compatibility to the elements of a basis of closed paths to ensure a compatible interval assignment.

Therefore, we first need to compute a basis of closed paths. In Appendix A, we present an original algorithm that computes the kernel of the matrix $\mathbf{B}$ defined in (6). For instance, Figure 10 shows a basis of closed paths for a multiply connected volume. This basis is computed using Algorithm 4 (see Appendix). Note that, in this example, the last path is not defined by the contour of a single face.

**Proposition 1.** ***Dimension of closed paths.*** *The dimension, $s$, of the kernel of closed paths is:*

$$s := \dim(\ker \partial) = \dim \mathbb{E} - \dim \mathbb{V} + 1 = q - p + 1. \tag{8}$$

*The proof of this Proposition is detailed in Appendix B.*

Once a basis of closed paths is computed we have to classify every edge, $e$, of every element, $\omega$, of the basis of closed paths. This classification is performed

in the computational domain. Given an edge, $e$, it is classified according to the direction it is traveled in the path $\omega$. Thus, an edge can be classified as $I_\omega^\pm$, $J_\omega^\pm$ or $K_\omega^\pm$. Then, each closed path defines the following compatibility equations:

$$\begin{array}{rcl} \displaystyle\sum_{e\in I_\omega^+} n_e & = & \displaystyle\sum_{e\in I_\omega^-} n_e, \\ \displaystyle\sum_{e\in J_\omega^+} n_e & = & \displaystyle\sum_{e\in J_\omega^-} n_e, \\ \displaystyle\sum_{e\in K_\omega^+} n_e & = & \displaystyle\sum_{e\in K_\omega^-} n_e. \end{array} \tag{9}$$

**Proposition 2.** *Let $(n_e)$ be a solution that verifies constraints* (9) *for a given basis of closed paths $\{\omega_1, \ldots, \omega_s\}$. Then, $(n_e)$ is a compatible number of intervals for every closed path of edges.*

*Proof.* Let

$$\omega = \sum_{i=1}^{s} \lambda_i \omega_i \tag{10}$$

be an arbitrary closed path. We have to prove that:

$$\begin{array}{l} \displaystyle\sum_{e\in I_\omega^+} n_e - \sum_{e\in I_\omega^-} n_e = 0, \\ \displaystyle\sum_{e\in J_\omega^+} n_e - \sum_{e\in J_\omega^-} n_e = 0, \\ \displaystyle\sum_{e\in K_\omega^+} n_e - \sum_{e\in K_\omega^-} n_e = 0. \end{array}$$

For the sake of simplicity, we will only prove the first of the three equalities. Since $(n_e)$ satisfies constraints (9) for every element, $\omega_i$, of the basis of closed paths, it verifies:

$$\begin{array}{rcl} \displaystyle\sum_{e\in I_{\omega_i}^+} n_e & = & \displaystyle\sum_{e\in I_{\omega_i}^-} n_e, \\ \displaystyle\sum_{e\in J_{\omega_i}^+} n_e & = & \displaystyle\sum_{e\in J_{\omega_i}^-} n_e, \\ \displaystyle\sum_{e\in K_{\omega_i}^+} n_e & = & \displaystyle\sum_{e\in K_{\omega_i}^-} n_e. \end{array}$$

We split the edges of the basis $\{\omega_1, \ldots, \omega_s\}$ in two sets according to the sign of coefficients $\{\lambda_1, \ldots, \lambda_s\}$ in Equation (10). That is, we define the sets

$$\left\{ \begin{array}{l} \Lambda^+ = \{i | \lambda_i \geq 0\}, \\ \Lambda^- = \{i | \lambda_i < 0\}. \end{array} \right. \tag{11}$$

The set of edges that are present in $I^+_\omega$ can be expressed in terms of the edges of the basis of close paths in the $I$ direction. According to the classification introduced in (11), $I^+_\omega$ can be constructed from the following two contributions.

$$I^+_\omega = \underbrace{\sum_{i\in\Lambda^+} \lambda_i I^+_{\omega_i}}_{\substack{\text{The } I^+_{\omega_i} \text{ edges that are} \\ \text{traveled in the } I^+ \text{ direction in } \omega \\ \text{(since } \lambda_i \text{ is positive)}}} + \underbrace{\sum_{i\in\Lambda^-} \lambda_i I^-_{\omega_i}}_{\substack{\text{The } I^-_{\omega_i} \text{ edges that are} \\ \text{traveled in the } I^+ \text{ direction in } \omega \\ \text{(since } \lambda_i \text{ is negative)}}} .$$

Similarly, the set $I^-_\omega$ can be constructed from the following two contributions.

$$I^-_\omega = \underbrace{\sum_{i\in\Lambda^+} \lambda_i I^-_{\omega_i}}_{\substack{\text{The } I^-_{\omega_i} \text{ edges that are} \\ \text{traveled in the } I^- \text{ direction in } \omega \\ \text{(since } \lambda_i \text{ is positive)}}} + \underbrace{\sum_{i\in\Lambda^-} \lambda_i I^+_{\omega_i}}_{\substack{\text{The } I^+_{\omega_i} \text{ edges that are} \\ \text{traveled in the } I^- \text{ direction in } \omega \\ \text{(since } \lambda_i \text{ is negative)}}} .$$

Therefore,

$$\sum_{e\in I^+_\omega} n_e = \sum_{i\in\Lambda^+} \lambda_i \left( \sum_{e\in I^+_{\omega_i}} n_e \right) - \sum_{i\in\Lambda^-} \lambda_i \left( \sum_{e\in I^-_{\omega_i}} n_e \right), \tag{12}$$

$$\sum_{e\in I^-_\omega} n_e = \sum_{i\in\Lambda^+} \lambda_i \left( \sum_{e\in I^-_{\omega_i}} n_e \right) - \sum_{i\in\Lambda^-} \lambda_i \left( \sum_{e\in I^+_{\omega_i}} n_e \right). \tag{13}$$

Subtracting equations (12) from (13),

$$\sum_{e\in I^+_\omega} n_e - \sum_{e\in I^-_\omega} n_e =$$

$$\sum_{i\in\Lambda^+} \lambda_i \left( \sum_{e\in I^+_{\omega_i}} n_e - \sum_{e\in I^-_{\omega_i}} n_e \right) - \sum_{i\in\Lambda^-} \lambda_i \left( \sum_{e\in I^+_{\omega_i}} n_e - \sum_{e\in I^-_{\omega_i}} n_e \right) =$$

$$\sum_{i\in\Lambda^+} |\lambda_i| \left( \sum_{e\in I^+_{\omega_i}} n_e - \sum_{e\in I^-_{\omega_i}} n_e \right) + \sum_{i\in\Lambda^-} |\lambda_i| \left( \sum_{e\in I^+_{\omega_i}} n_e - \sum_{e\in I^-_{\omega_i}} n_e \right) =$$

$$\sum_{i=1}^{s} |\lambda_i| \underbrace{\left( \sum_{e\in I^+_{\omega_i}} n_e - \sum_{e\in I^-_{\omega_i}} n_e \right)}_{=0} = 0.$$

□

Proposition 2 ensures that a solution that verifies constraints (9) for a given basis of closed paths generates a valid interval assignment in the whole volume. Therefore, we modify the integer linear problem (4) in order to take into account equations (9). The new integer problem to solve is:

$$\begin{cases} \min \sum_{e} \omega_e (D_e + d_e) + M - m, & \\ \text{constrained to:} & \\ \sum_{e \in I_\omega^+} n_e = \sum_{e \in I_\omega^-} n_e, & \text{for each path } \omega, \\ \sum_{e \in J_\omega^+} n_e = \sum_{e \in J_\omega^-} n_e, & \text{for each path } \omega, \\ \sum_{e \in K_\omega^+} n_e = \sum_{e \in K_\omega^-} n_e, & \text{for each path } \omega, \\ D_e - d_e = n_e - N_e, & \text{for all edges,} \\ D_e, d_e \geq 0, & \text{for all edges,} \\ M \geq n_e - N_e, & \text{for all edges,} \\ m \leq n_e - N_e, & \text{for all edges,} \end{cases} \tag{14}$$

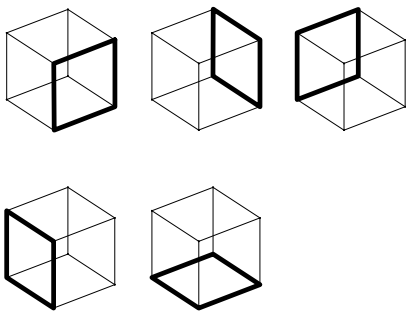


Figure 11: Basis of closed paths for a cubic geometry.

The solution of this problem provides a number of edge divisions such that a structured hexahedral mesh can be generated inside the volume. It is important to point out that even if a volume does not contain any through holes, the proposed procedure still provides a basis of closed paths that ensures the mesh compatibility. Moreover, from the computational point of view, in these cases it is preferable to solve the integer linear problem (14) instead of the original integer linear problem (4) because the former contains less constraints. To illustrate this situation, Figure 11 presents a simple cube geometry where the five elements of the basis of closed paths are highlighted with thick solid lines. Note that the upper face does

not generate a closed path that belongs to the basis. Therefore, there are only five faces that contribute to the integer linear problem (14). However, there are six faces that contribute to integer linear problem (4). Therefore, we solve integer linear problem (14) whether the volume contains through holes or not. Algorithm 3 presents the general submapping method to mesh volumes that may contain inner or through holes. In this algorithm, multiply connected faces have to be converted into simply connected ones. To this end, we use the algorithm presented in [24].

**Algorithm 3** General submapping algorithm

**Ensure:** aMesh

```
1:  function SUBMAPPING(Volume)
2:      Vertex classification
3:      Edge classification
4:      Face classification
5:      for all face do
6:          if face is multiply connected then
7:              convert the face to simply connected
8:          end if
9:      end for
10:     Computation of a basis of closed paths and associated constraints
11:     Interval assignment: solution of ILP (14)
12:     Boundary discretization
13:     if Volume has inner holes then
14:         Connect inner boundaries with outer boundary
15:     end if
16:     Geometry decomposition
17:     Patch discretization
18:     if Volume has inner holes then
19:         Mesh the volume defined by the virtual surfaces
20:     end if
21: end function
```

# 4 Examples

This section presents six examples of meshes generated using the proposed algorithm for the submapping method. The user assigns a global element size and, optionally, a fixed or minimum number of intervals to some edges. Then, the algorithm presented in Section 3.2 automatically detects the necessary constraints to assign a compatible number of intervals. Note that the new integer linear problem

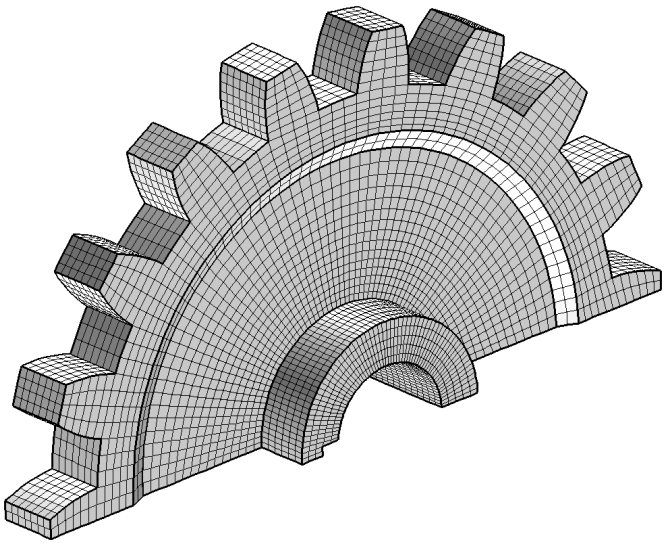


Figure 12: Mesh generated for the half of a gear using the proposed method. The model does not contain inner voids nor through holes. The mesh is composed of 3064 nodes and 1989 hexahedra.

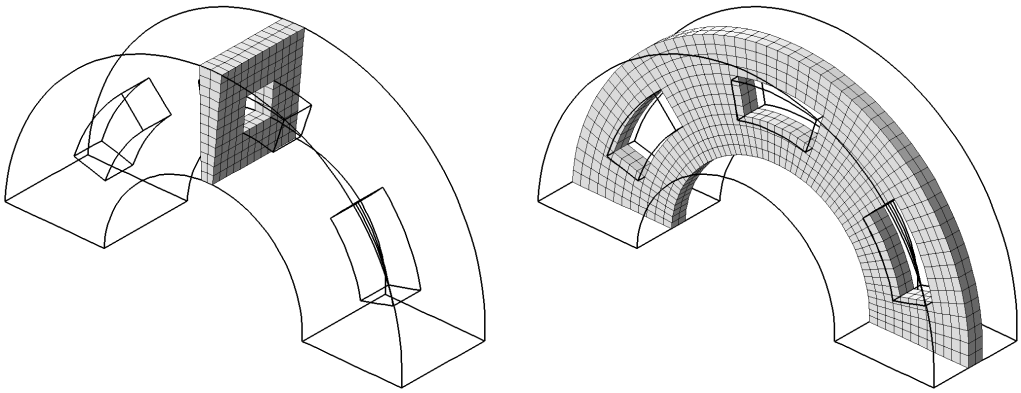
(14) is solved, even if through holes are not present. In addition, the technique presented in Section 3.1 automatically connects inner voids with the outer boundary. Finally, the geometry is decomposed and classical mapping methods are applied.

## 4.1 Half of a gear

The objective of the first example is to verify that the proposed algorithm is able to discretize simply connected volumes. That is, we will show that the new integer linear problem (14) is able to assign a compatible number of intervals for simply connected geometries. Since the volume does not contain any inner hole, it is not necessary to apply the algorithm to connect inner voids with the outer boundary. Figure 12 shows the mesh generated using the proposed algorithm for a half of a gear.

## 4.2 Inner holes

The second example is devoted to the discretization of a geometry with inner holes. In this case, it is necessary to apply the proposed automatic algorithm presented in section 3.1 to convert the volume into simply connected to mesh it using submapping. Figure 13 shows two cross sections of the generated mesh



(a) (b)

Figure 13: Mesh generated for the geometry of the second example. The model only contains inner voids. The mesh is composed of 4.228 nodes and 3.216 hexahedra. (a)-(b) Cross sections of the mesh generated by the proposed method.

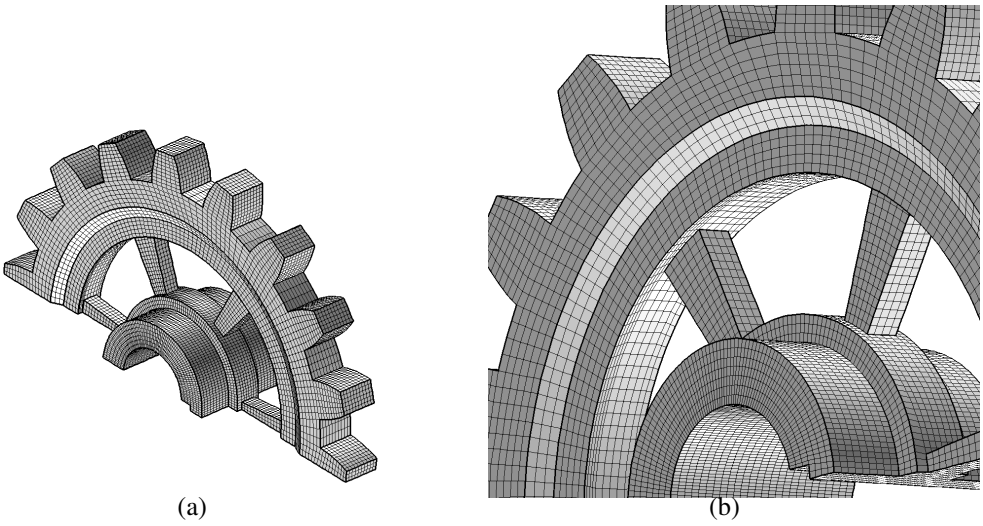
using the proposed algorithm.

## 4.3 Light gear

The third example presents the discretization of a light gear. In this case, the geometry contains through holes. Therefore, we have to apply the algorithm presented in section 3.2 in order to assign a compatible number of intervals. Since the geometry does not have inner holes, we do not need to connect inner boundaries to the outer boundary. Figure 14(a) presents the final mesh generated using the proposed method and Figure 14(b) shows a detail of the obtained mesh.

## 4.4 Bench

The fourth example is devoted to the discretization of a bench. In this case, the geometry contains a large number of through holes, and it is composed by a large number of components. In addition, it is important to point out that the through holes are oriented in several directions. Hence, our algorithm to assign the number of intervals is crucial to generate a high quality mesh. Figure 15(a) presents a general view of the mesh generated by the proposed method in the bench model, while Figures 15(b) and 15(c) present two details of the final mesh.



(a) (b)

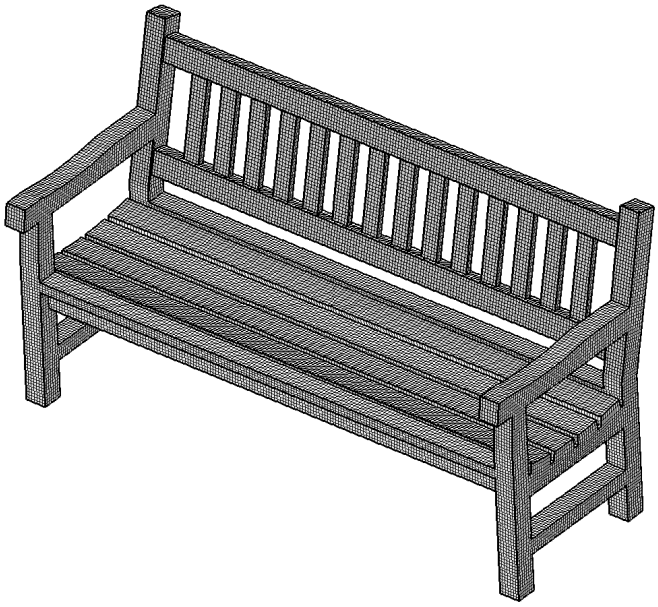
Figure 14: Mesh generated for a gear model using the proposed method. The model only contains through holes. The mesh is composed of 43.970 nodes and 36.125 hexahedra. (a) General view of the mesh. (b) A detailed view of the mesh.

## 4.5 Room with two columns

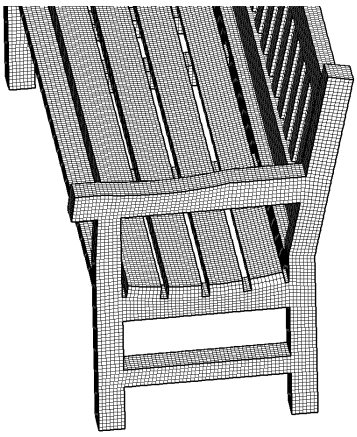
The fifth example is devoted to the discretization of a volume that contains both inner and through holes. Figure 16 presents a model of a room with two columns. In this example, we proceed to mesh the walls and the columns. To this end, we have to apply the algorithms developed in Section 3.1 and 3.2. In addition, we imposed a higher mesh density in the vertical direction. The proposed algorithm successfully detects the necessary constraints in order to compute a compatible interval assignment. Then, the inner boundary is automatically connected to the outer boundary in order to convert the volume into simply connected. Figure 17 presents a detail of the mesh generated using the proposed algorithm.

## 4.6 Mechanical piece

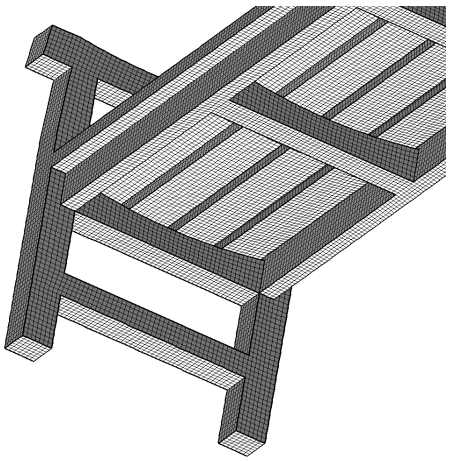
The sixth example presents the mesh generated on a mechanical piece, see Figure 18. This geometry contains fillets and circular-shaped through holes. Hence, this volume is not automatically meshed because the classification of vertices and edges do not produce a valid computational domain. However, our algorithm allows to manually prescribe the edge and vertex classifications. Then, the algorithm proceeds to compute a basis of closed paths to ensure a valid interval assignment.



(a)



(b)



(c)

Figure 15: Mesh generated for a bench model using the proposed method. The model only contains through holes. The mesh is composed of 110.739 nodes and 80.062 hexahedra. (a) General view of the mesh. (b) Detail of the armrest. (c) Detail of the bottom zone.

## 4.7 Summary

Table 1 presents the information of the solved ILP's for each of the examples. In the examples that do not contain through holes (gear and example 2), there are

more faces than closed paths in the basis. On the contrary, when the geometry contains through holes, the number of closed paths is smaller that the number of faces. The total time to solve the ILP's is almost negligible for all the examples. The most expensive ILP took two seconds, approximately. All the examples were

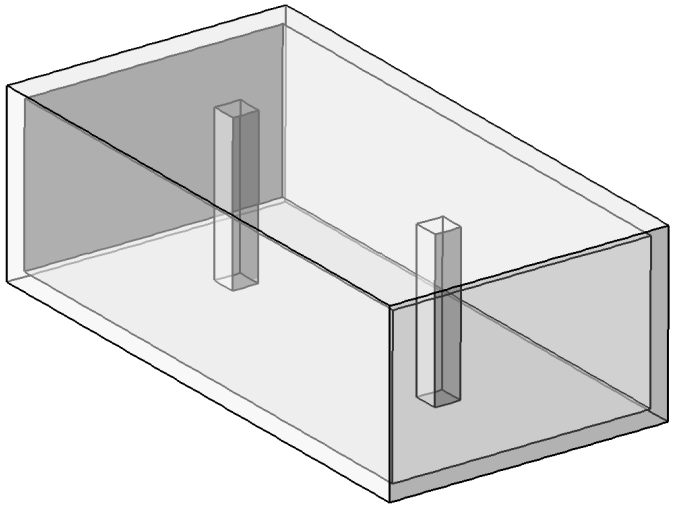


Figure 16: CAD model of a room with two columns. The model contains inner voids and through holes.

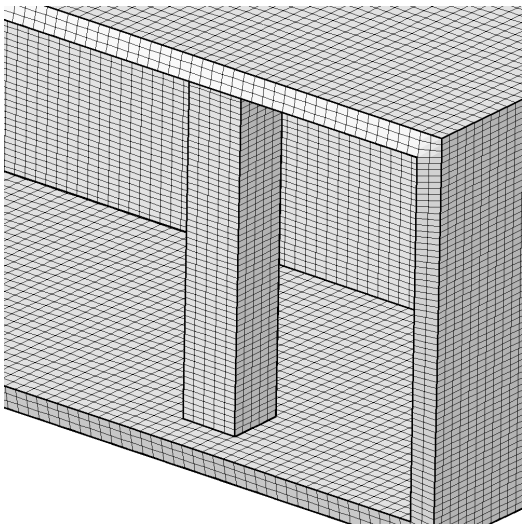
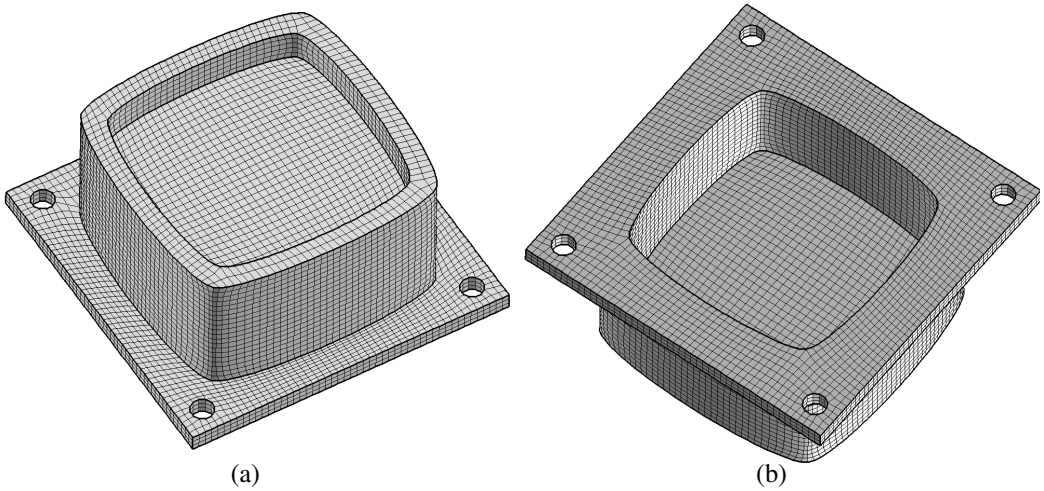


Figure 17: Detail of the mesh generated for the walls of a room. The mesh is composed of 60.663 nodes and 39.444 hexahedra.



(a) (b)

Figure 18: Mesh generated for a mechanical piece using the presented method. The model only contains through holes. The mesh is composed of 19.720 nodes and 14.092 hexahedra. (a)-(b) Two different view of the generated mesh.

Table 1: Number of edges, faces and size of the solved ILP's for the geometries of the presented examples.

| geometry | edges | faces | closed paths | equations | variables | time (s) |
|---|---|---|---|---|---|---|
| gear | 153 | 53 | 52 | 563 | 461 | 0.329 |
| example 2 | 48 | 32 | 20 | 184 | 146 | 0.078 |
| light gear | 237 | 79 | 84 | 883 | 713 | 0.594 |
| bench | 659 | 191 | 260 | 2548 | 1979 | 1.968 |
| room | 56 | 20 | 22 | 214 | 170 | 0.078 |
| mechanical piece | 129 | 37 | 44 | 479 | 389 | 0.283 |

executed on a Pentium Core 2 Duo CPU at 3.00 Mghz with 2GB RAM.

# 5 Conclusions

In this work we have presented three original contributions in order to improve the applicability of the submapping method. The first one is focused on the ability to obtain high quality interval assignment. The meshes obtained by the submapping method are heavily affected by the solution of an integer linear problem. In fact, the quality of the interval assignment is governed by the objective function that defines the integer linear problem. Hence, we have proposed a new objective function in order to take into account the maximum and minimum differences be-

tween the target number of intervals prescribed by the user and the actual number of intervals. Although we have only presented five examples, the proposed objective function has been successful used in dozens of models. In all cases it better distributes the elements between the edges of the geometry.

The second contribution is focused on the discretization of volumes that contain inner voids. To this end, the proposed automatic procedure connects inner boundaries with the outer boundary using virtual surfaces. Virtual surfaces are computed using an auxiliary constrained Delaunay tetrahedralization of the volume. Once all the holes are connected, the classical submapping method is applied in order to obtain the final mesh. It is important to point out that the CPU time involved in this procedure is not relevant since the Delaunay tetrahedralization is performed using only the surface meshes.

The third contribution deals with the discretization of volumes that contain through holes. When through holes are present, additional constraints have to be imposed in the integer linear problem (4). The basic idea of the proposed algorithm is to impose a compatible number of intervals to the elements of a basis of closed path of edges. The proposed procedure automatically recognizes a basis of closed paths and builds the new integer linear problem (14). Once a compatible number of intervals has been computed, the domain is splitted and classical mapping algorithms are applied.

Finally, it is worth to notice that the poroposed algorithm for the submapping method presented in this work is successfully implemented in the ez4u meshing environment [25].

# Acknowledgements

This work was partially sponsored by the Spanish *Ministerio de Ciencia e Innovación* under grants DPI2007-62395, BIA2007-66965 and CGL2008-06003-C03-02/CLI.

# A Appendix

In this appendix, we present an original algorithm to compute the kernel of matrix $\mathbf{B}_{(p\times q)}$, where $p$ and $q$ are the number of vertices and edges of the geometry, respectively. The resulting basis of the kernel of matrix $\mathbf{B}$ is stored in the columns of a rectangular matrix $\mathbf{W}_{(q\times s)}$, where $s$ is the number of elements in the basis. Note that the number of elements of the basis is known a priori, see Equation (8). In this algorithm, we denote by $\mathbf{B}_k$ the $k$-th column of matrix $\mathbf{B}$. Two additional matrices will be used, $\mathbf{B}'_{(p\times q)}$ and $\mathbf{K}_{(q\times q)}$. In the algorithm, matrices $\mathbf{B}'$ and $\mathbf{K}$ are

modified in such a way that the following equation is verified during the whole algorithm

$$\mathbf{BK} = \mathbf{B}'. \tag{15}$$

Matrices $\mathbf{B}'$ and $\mathbf{K}$ are initialized as $\mathbf{B}$ and $\mathbf{I}_{(q\times q)}$, respectively. The objective of the algorithm is to transform matrix $\mathbf{B}'$ in such a way that it contains $s$ null columns. Note that if the $k$-th column of $\mathbf{B}'$ is null, then the $k$-th column of $\mathbf{K}$ belongs to the kernel of $\mathbf{B}$ since equation (15) is verified during the whole algorithm.

**Algorithm 4** Kernel

**Ensure:** $\mathbf{W} = Ker(\mathbf{B})$

**function** KERNEL($\mathbf{B}$)
  $W \leftarrow \{\}$
  $\mathbf{K} \leftarrow \mathbf{I}_{(m\times m)}$
  **for** $k \leftarrow 1 : q$ **do**
    **if** $\mathbf{B}_k = \mathbf{0}$ **then**
      append $\mathbf{K}_k$ in $\mathbf{W}$
    **else**
      $i \leftarrow 1$
      **while** $b_{ik} = 0$ **do**
        $i \leftarrow i + 1$
      **end while**
      **for** $j \leftarrow k + 1 : q$ **do**
        **if** $b_{ij} \neq 0$ **then**
          $\mathbf{B}_j \leftarrow \mathbf{B}_j - b_{ij}b_{ik}\mathbf{B}_k$
          $\mathbf{K}_j \leftarrow \mathbf{K}_j - b_{ij}b_{ik}\mathbf{K}_k$
        **end if**
      **end for**
    **end if**
  **end for**
**end function**

To generate a null column in $\mathbf{B}'$, the algorithm modifies the columns of $\mathbf{B}'$ by computing linear combinations of its columns. We modify $\mathbf{K}$ accordingly, in order to satisfy equation (15). In fact, the actual algorithm does not use the auxiliary matrix $\mathbf{B}'$. Instead, all the operations are performed on matrix $\mathbf{B}$, in order to save computer memory. Algorithm 4 details the proposed method to compute a basis of the kernel of $\mathbf{B}$.

# B Appendix

In this Appendix we detail the proof of Proposition 1. That is, we prove that $s := \dim(\ker \partial) = \dim \mathbb{E} - \dim \mathbb{V} + 1 = q - p + 1$, where $\partial$ is the boundary linear operator introduced in equation (5), and $p$ and $q$ are the number of vertices and edges of the geometry, respectively.

*Proof.* Consider the following chain of applications:

$$\{0\} \xrightarrow{\partial_2} \mathbb{E} \xrightarrow{\partial_1} \mathbb{V} \xrightarrow{\partial_0} \{0\}$$

where: $\partial_2(0) := \mathbf{0}$; $\partial_1 := \partial$, that is, $\partial_1$ is the boundary linear operator introduced in Equation (5); and $\partial_0(\mathbf{v}) := 0,\ \forall \mathbf{v} \in \mathbb{V}$. According to these definitions, it is straightforward to prove that $\operatorname{Im} \partial_{i+1} \subseteq \ker \partial_i$ for $i = 0, 1$. We define the following quotient groups as:

$$K_i = \frac{\ker \partial_i}{\operatorname{Im} \partial_{i+1}}, \quad i = 0, 1.$$

On the one hand, the Euler's characteristic of this chain is (see reference [26] for more details about the Euler characteristic)

$$\chi = \dim \mathbb{V} - \dim \mathbb{E} = p - q. \tag{16}$$

On the other hand, the Euler's characteristic also verifies

$$\chi = \dim K_0 - \dim K_1. \tag{17}$$

It is well known that $\dim K_0$ equals to the number of connected components of the graph, see [26]. In this case, $\dim K_0 = 1$.

In addition, we have that $\dim K_1 = \dim(\ker \partial_1) - \underbrace{\dim(\operatorname{Im} \partial_2)}_{=0} = \dim(\ker \partial_1)$.

Therefore, equation (17) becomes

$$\chi = 1 - \dim(\ker \partial_1). \tag{18}$$

Finally, using equations (16) and (18), we obtain that:

$$\dim(\ker \partial_1) = q - p + 1.$$

□

# References


[1] Thompson JF, Soni B, Weatherill N, Handbook of Grid Generation, CRC Press, 1999.

[2] Frey PJ, George PL, Mesh generation: application to finite elements, Wiley, 2008.

[3] Schneiders R A grid-based algorithm for the generation of hexahedral element meshes, Eng Comput, 1996; 12:168-177.

[4] Taghavi R, Automatic, parallel and fault tolerant mesh generation from CAD, Eng Comput, 1996; 12:178-185.

[5] Price M, Armstrong C, Hexahedral mesh generation by medial surface subdivision: Part I, Int J Numer Meth Eng, 1995; 38:3335-3359.

[6] Price M, Armstrong C, Hexahedral mesh generation by medial surface subdivision: Part II, Int J Numer Meth Eng, 1997; 40:111-136.

[7] Staten M, Kerr R, Owen S, Blacker T, Unconstrained paving and plastering:progress update, 6th International Meshing Roundtable, 2006; Sandia National Laboratories, pp. 195-196.

[8] Tautges TJ, Blacker T, Mitchell SA, The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes, Int J Numer Meth Eng, 1996; 39:3327-3349.

[9] Calvo NA, Idelsohn SR, All-hexahedral element meshing: Generation of the dual mesh by recurrent subdivision, Comput Methods Appl Mech Eng, 2000; 182:371-378.

[10] Cook WA, Oakes WR, Mapping methods for generating three-dimensional meshes, Comp Mech Eng, 1982; 8:67-72,

[11] White D, Automatic Quadrilateral and hexahedral meshing of pseudo-cartesian geometries using virtual subdivision, Master thesis, Brigham Young University, 1996.

[12] Whiteley M, White D, Benzley S, Blacker T, Two and three-quarter dimensional meshing facilitators, Eng Comput, 1996; 12:144-154.

[13] Knupp PM, Next-Generation Sweep Tool: A Method For Generating All-Hex Meshes On Two-And-One-Half Dimensional Geomtries, 7th International Meshing Roundtable, 1998; Sandia National Laboratories, pp: 505-513.

[14] Blacker T, The Cooper Tool, 5th International Meshing Roundtable, 1996; Sandia National Laboratories, pp: 13-30.

[15] Staten ML, Canann SA, Owen, SJ, BMSweep: Locating interior nodes during sweeping, Eng Comput, 1999; 15:212-218.

[16] Roca X, Sarrate J, An automatic and general least-squares projection procedure for sweep meshing, Eng Comput, Accepted for publication.

[17] Roca X, Sarrate J, A new least-squares approximation of affine mappings for sweep algorithms, Eng Comput, Accepted for publication.

[18] Mitchell SA, High fidelity interval assignment, Int J Comput Geom Ap 2000; 10:399-415.

[19] Schrijver A, Theory of Linear and Integer Programming, John Wiley and Sons, 1998.

[20] http://sourceforge.net/projects/lpsolve

[21] Open cascade technology, 3d modeling & numerical simulation, http://www.opencascade.org

[22] Si H, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, http://tetgen.berlios.de

[23] Deo N, Graph theory with its application to engineering, Prentice-Hall, 1990.

[24] Ruiz-Gironés E, Sarrate J, Generation of structured meshes in multiply connected surfaces using submapping, Adv Eng Softw 2010; 41:379-387.

[25] Roca X, Sarrate J, Ruiz-Gironés E, A graphical modeling and mesh generation environment for simulations based on boundary representation data, Congresso de Métodos Numéricos em Engenharia, Porto, Portugal, 2007.

[26] Hatcher A, Algebraic topology, Cambridge University Press, 2002.